Patterns of Flexible Modeling Tools

FILIPE FIGUEIREDO CORREIA, Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto
ADEMAR AGUIAR, INESC Porto, Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto

The benefits of using models have long been acknowledged by research and industry, but in practice the use of modeling tools often implies a more-than-reasonable effort or confines itself to points in the project lifetime when the requirements and/or design is well understood. Free-form tools like whiteboards and textual documents fill information capture needs during the rest of the time — they pose a lower barrier for adoption and enable users to capture their flow of thought with less constraints than a formal modeling tool would allow.

Flexible Modeling Tools try to provide a compromise of both approaches. The works discussed at the FlexiTools workshop series represent an interesting body of knowledge, issues and approaches for this class of tools, and was one of the main sources for mining the patterns in this paper — USER-CRAFTED STATIC META-MODEL, MODEL CO-EVOLUTION, META-MODELING BY EXAMPLE, FORMALIZATION, LINKED MODELS and AUGMENTED MODELS. These patterns identify several approaches that can be used by those developing modeling tools with flexibility requirements. They don't exhaust all the approaches in the area, but intend to represent the most relevant ones.

Categories and Subject Descriptors: D.2.11 [**Software Architectures**] Patterns

General Terms: Flexible Modeling

ACM Reference Format:

Correia, F. F. and Aguiar, A. 2013. Patterns of Flexible Modeling. 20th Conference on Pattern Languages of Programs (PLoP) – Monticello, Illinois, USA (October 2013), 17 pages.

1. INTRODUCTION

To create a model is to represent a complex reality in simple terms, focusing on capturing only its relevant aspects. Meta-modeling, in turn, is the use of models to specify other models — it comprises the *analysis, construction and development of the frames, rules and constraints to modeling a predefined class of problems* [Stahl and Voelter 2006].

But more than two meta-modeling levels can be considered. As an example, the UML (Unified Modeling Language), one of the most widespread modeling languages, is based on the Meta Object Facility (MOF) [OMG – MOF], a meta-modeling architecture consisting of four modeling levels, each conforming to the one above — M0, M1, M2 and M3, with M0 corresponding to the *data*, M1 to the *model*, M2 to the *meta-model* and M3 to the *meta-meta-model*, which is compliant with itself. An example of these layers of abstraction are depicted in Figure 1.

The frames, rules and constraints mentioned above can be regarded as the *structure* of the contents. By formally (i.e., explicitly) capturing this structure, modelers are expressing information with a higher degree of rigor and

This work was partially supported by Fundação para a Ciência e Tecnologia and by ParadigmaXis, through the grant SFRH/BDE/33883/2009.
Authors' addresses: F. F. Correia, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal; email: filipe.correia@fe.up.pt; A. Aguiar, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal; email: ademar.aguiar@fe.up.pt

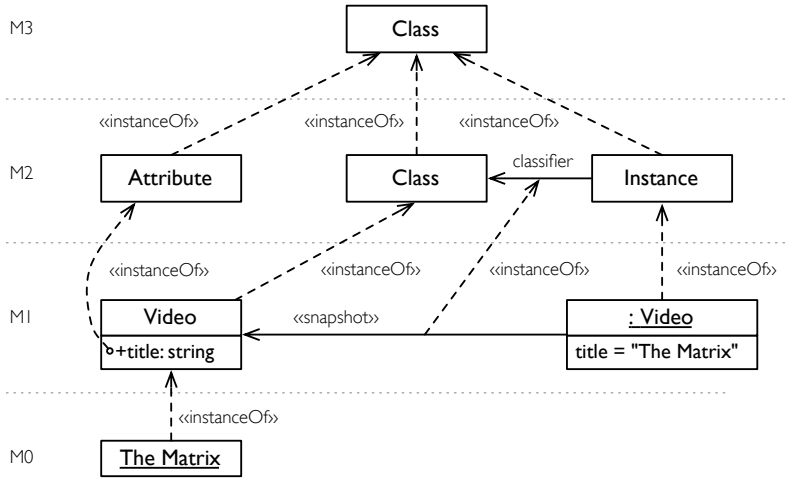

Fig. 1. MOF's abstraction layers, adapted from *Adaptive Object-Models: a Research Roadmap* [Ferreira et al. 2010].

objectivity than what would be achieved by capturing free-form contents. Not only is this an advantage from an expressiveness standpoint, it allows to automatically process information in multiple ways, like ensuring its consistency or reusing it in different contexts.

But, despite their benefits, modeling tools are still often dismissed in favor of lighter weight approaches, that are easier to learn and free users from obeying a model's constraints. Such approaches range from textual documents to paper drawings, to white board sketches. The freedom that they allow is especially important during exploratory phases of requirements engineering and design, when creativity and the ability to record incomplete information becomes more important that rigor. The ultimate goal, however, is always to achieve the rigor and objectivity required to build a software system, so it's ironic that the enforcement of the rules and constraints, that is inherent to most modeling tools, is also the reason why they are so cumbersome to use when creativity and exploratory design is called for.

Flexible Modeling Tools strive to combine the advantages of both approaches, allowing users to trade precision for flexibility, and vice-versa, whenever the occasion calls for it. In the next section you will find an overview of six patterns, and Sections 3 to 8 cover the patterns themselves.

2. ABOUT THE PATTERNS

These patterns are addressed to software developers wanting to gain a better understanding of the trade-offs and approaches of creating modeling tools with flexibility goals. In other words, they can support the creation of modeling tools specifically designed to help their users' work without constraining them in undesirable ways. They may, for example, allow users[1] to pick just the right amount of structure for a given piece of information and allow to change it in the future, or allow to connect elements of a model to related external contents.

Meta-modeling is within the scope of these patterns as, a) meta-modeling is in itself a form of modeling, and b) modeling activities always imply the use of a meta-model, whether intrinsic and implicit (e.g., programmed; enforced by the tool alone) or extrinsic and explicit (e.g., represented using an XML dialect). Domain Specific

[1] We often refer to *users*, and it is perhaps worth clarifying that, in the context of this paper, they are the *modelers* — i.e., the end-users of the modeling tools.

Languages (DSLs) can also be considered within the scope of these patterns, as their creation and use are in fact meta-modeling and modeling activities using a specific type of representation. We have, however, deliberately not addressed any issue particular to how models may be represented (e.g., graphically or textually), as well as other concerns that are not specific to modeling *flexibility*. Additionally, flexibility is a function of how model and meta-model constraints are approached by developers, and the implications of that can span several different facets of a modeling tool, from user-interaction, to domain validations, to persistence. These patterns don't go into the details of how they may influence these facets of a modeling tool, but they could certainly be extended into a wider *pattern language* covering such concerns.

The *flexible modeling* principles described by these patterns are applicable to a wider scope than what may strictly be considered a *modeling tool*. Even if not always viewed in these terms, all representations of structured information may be regarded as a model, and the tools that allow to create and maintain them can be seen as tools supporting *modeling activities*. In this sense, these patterns may be useful to developers creating any sort of tool focused of manipulating representations of structured contents. Figure 2 depicts a map of the six patterns that will be introduced.

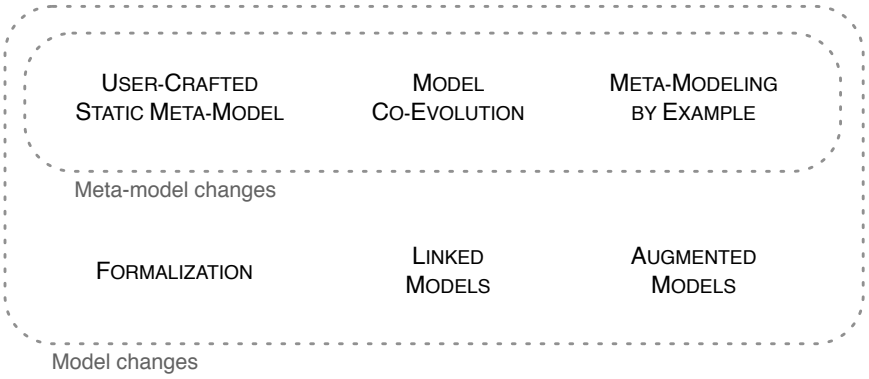

Fig. 2. Flexible Modeling Patterns Map grouped by the meta-level that is the focus of changes.

- User-Crafted Static Meta-Model (5) — Supports the creation of a meta-model by the modeler himself, before the model is built.
- Model Co-Evolution (6) — Explains how models can be evolved together with the meta-models that they depend on, when these meta-models need to change;
- Meta-Modeling by Example (8) — Allows to create higher abstraction levels (e.g., meta-models) by giving examples at lower levels (e.g., models).
- Formalization (10) — Supports deriving formal/rigorous representations (e.g., models) from information that was firstly captured in informal/non-rigorous ways;
- Linked Models (12) — Provides a way to connect different but complementary models flexibly;
- Augmented Models (14) — Supports complementing or annotating models with contextual information, that is often not structured or is loosely structured;

Figure 2 also shows, for each of the patterns, which meta-levels are subject to change. Other criteria can be used to group the patterns — Figure 3 depicts which patterns focus on relaxing or leveraging the constraints between

the model and meta-model levels, and which ones focus on doing the same for the relations between models and other, external, contents.

Even though the end goal of these patterns is mainly to support the creation of models, USER-CRAFTED STATIC META-MODEL, MODEL CO-EVOLUTION and META-MODELING BY EXAMPLE directly support creating or introducing changes at the meta-model level. As depicted by Figure 3, together with FORMALIZATION, they have in common the goal of providing flexibility between the two modeling levels.

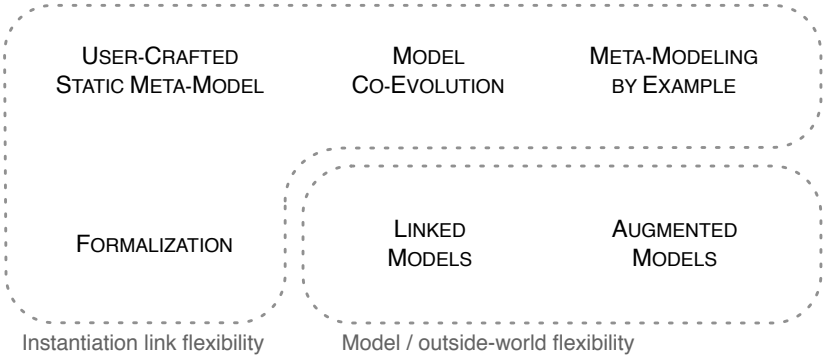

Fig. 3. Flexible Modeling Patterns Map grouped by the *link* that is the focus of flexibility.

FORMALIZATION, LINKED MODELS and AUGMENTED MODELS differ from the other three patterns in that they focus on the introduction of changes only at the model level. Finally, and not explicitly shown by the figures, LINKED MODELS can be said to be a form of supporting AUGMENTED MODELS, as the contents used to *augment* the model are, in fact, elements from another model.

These patterns were mined mainly from the body of works discussed at the latest three editions of the FlexiTools workshop series[2] [Ossher et al. 2010; Kimelman et al. 2010; Ossher et al. 2011]. Rather than adopting a specific view of what makes a modeling tool *flexible*, they try to reflect the current understanding of the area as taken by its community. Some works in particular have already tried to categorize challenges and approaches taken in this area, and influenced the recognition of these patterns — the very complete summary of the FlexiTools workshop at SPLASH 2010 [3] by Kimelman and Hirschman [Kimelman and Hirschman 2010], a work by Gabrysiak et al, that proposes a classification of meta-models' usage scenarios [Gabrysiak et al. 2011] and another work by Kimelman and Hirschman [Kimelman and Hirschman 2011] that includes different interpretations of the notion of modeling tool flexibility.

Some of the authors' previous experience have also influenced the identification of these patterns; namely, their work on Adaptive Object-models [Ferreira et al. 2008; Ferreira et al. 2010] and on Adaptive Software Artifacts [Correia 2010; Correia 2013].

[2]Workshop on Flexible Modeling Tools
[3]SPLASH 2010 — Systems Programming Languages and Applications: Software for Humanity

3. USER-CRAFTED STATIC META-MODEL

Modeling tools support their users in expressing information in a given domain or domains. The creation of a modeling tool must consider which domains the modeler will need to address and how expressive they will need to be on those domains.

3.1 Problem

The modeling tool developers may be unable to anticipate the domains and expressiveness that the modeler will need.

Conceiving modeling tools and their underlying meta-models requires good abstraction **skills** and considerable **effort**, making it compelling to **reuse** them rather than creating multiple ones, tailored to each specific context. Creating a modeling tool that adheres to a specific meta-model rather than supporting multiple ones also contributes to keeping the modeling tool **simple**.

On the other hand, if the abstractions provided by a modeling tool and its underlying meta-model are not suited for the intended domain, the user might not be sufficiently **expressive** in that domain. In such a case, using that meta-model might be impossible or imply more **effort**, and the resulting model might not always be **reliable**.

3.2 Solution

Allow the meta-model to be refined by the modeler, before a model is built.

Use this pattern when the user will need to model domains that can't be fully anticipated by the modeling tool developers. Instead of developing a modeling tool that uses only one specific meta-model and addresses a specific domain, allow the user to supply her own meta-model, tailored to a domain's needs. This means that modeling tool developers must resource to a meta-meta-model to define which meta-models may be supplied by the user. Often, the modeling tool may itself support the creation of the meta-models, in which case it will have a simpler design if the same mechanism is used to support the creation of both modeling levels (i.e., if EVERYTHING IS A THING [Ferreira et al. 2010]).

This pattern can also be referred to as *User-Generated Metamodel* [Gabrysiak et al. 2011].

3.3 Known Uses

USER-CRAFTED STATIC META-MODEL is perhaps the *least flexible* of this set of patterns and can be found on some *traditional* modeling tools. For example, the Eclipse Modeling Framework (EMF) supports defining a meta-model and, together with the GMP framework (Graphical Modeling Project), it allows to define a graphical representation and build a complete modeling tool for that meta-model.

MetaEdit+ [Tolvanen et al. 2007] is an environment that allows to create new modeling languages. It uses graphical meta-modeling to support the early stages of language creation, to define the key language concepts and rules. The meta-modeling language is defined as one of the several domain-specific languages supported by the platform.

3.4 Related Patterns

USER-CRAFTED STATIC META-MODEL differs from the other patterns described in this paper because the flexibility that it allows is only possible until the meta-model is instanced into a concrete model. Namely, it doesn't address meta-model changes after a model has been created. When modelers need the meta-model to evolve after it has been instanced, they need support for MODEL CO-EVOLUTION or META-MODELING BY EXAMPLE.

4. MODEL CO-EVOLUTION

USER-CRAFTED STATIC META-MODELS are created before modeling activities take place, and they allow modelers to define how expressive they will be able to be during such activities. To use that pattern effectively, modelers must have a very concrete idea of the domain that they will address *before* starting to model in that domain, as the modeling tool might not easily allow to refine that idea after the initial creation of a meta-model takes place.

4.1 Problem

Modelers may be unable to anticipate the expressiveness that they will need.

Models, and their semantics, directly depend on the constructs defined by the associated meta-models. Users may need to **change** the meta-model during the creation of a model, but they also want models' **consistency** towards their meta-model to be kept. Modelers don't want to spend a lot of **effort** manually maintaining this consistency if they can avoid it. Modeling tools often help by preventing some operations that would cause inconsistencies, but this may be limiting, as it means that the meta-model cannot be be **freely evolved** to all states that it once could, before there were models that used it.

4.2 Solution

Allow to change meta-models and automatically evolve dependent models accordingly.

Support MODEL CO-EVOLUTION when meta-models need to evolve even though models that are based upon them have already been created and the user needs these models to be kept in-sync with their changing meta-models.

Often the operations that are made available to the user at both levels — model and meta-model — are defined by a set of distinct classes that confines the changes that the modeling tool supports, thus applying the COMMAND pattern [Gamma et al. 1995].

To make the changes at the *meta-model level* have the right repercussions at the *model level*, operations executed at the meta-model level will spawn the execution of other operations at the model level. For each kind of meta-model change, the tool must know how to make the consequential changes at the model level, and sometimes it may require the modeler to provide additional instructions on how the model should be transformed.

The key principles of this pattern can also be applied between the *data* and *model* levels — changes to the model can trigger changes to any data that complies to it.

4.3 Known Uses

Gabrysiak et al emphasize the need for modeling flexibility and propose a tool that combines a) a minimization of the restrictions imposed by the meta-model to the model with b) the co-evolution of models as a result of introducing changes to their meta-models [Gabrysiak et al. 2010; Gabrysiak et al. 2011].

Some approaches and tools to support the evolution of models upon the introduction of changes to their meta-model have been proposed, including Wachsmuth's [Wachsmuth 2007] and Cicchetti's [Cicchetti et al. 2008] works.

Some object-oriented frameworks like RubyOnRails and Django support the creation of a domain model, that is persisted using an ORM. The creation of this domain model usually doesn't involve the use of a modeling tool, nor is its metamodel subject to changes, but the model can, and often does, change during the development of a system. Existing data that complies with such models needs to be changed accordingly, and these frameworks very often provide the mechanisms to do so (e.g., *Migrations* in the case of RoR and *South* in the case of Django).

4.4 Related Patterns

Like META-MODELING BY EXAMPLE, and unlike USER-CRAFTED STATIC META-MODEL, MODEL CO-EVOLUTION supports evolving the meta-model after it is instanced.

Co-Evolution is a general concept, that may be applied to other domains. Namely, it is often used in the context of software documentation as described by the CO-EVOLUTION pattern [Correia et al. 2009].

As mentioned above, this pattern is also applicable between the model and data levels. In particular, the MIGRATION pattern [Ferreira et al. 2008] supports co-evolution between a model and existing data that complies to that model, in the context of ADAPTIVE OBJECT-MODELS [Ferreira et al. 2009].

5. META-MODELING BY EXAMPLE

Both modeling tool developers and the modelers themselves may be unable to anticipate the expressiveness that the modelers will need. To use USER-CRAFTED STATIC META-MODELS effectively, modelers must have very concrete ideas of the domain that they will address *before* starting to model in that domain. MODEL CO-EVOLUTION opens the possibility to refine those ideas after the initial creation of a meta-model takes place, but modelers often discover the new directions that the meta-model will take when trying to express a model and exploring their options at that abstraction level.

5.1 Problem

The need to change the meta-model diverts the modeler from the creation of the model.

The need to constantly update the meta-model to allow for more expressiveness at the model level diverts the modeler from her main **stream of thought**. Moreover, the need to create or update a meta-model is often a **barrier to entry**, especially when modelers don't have the required **higher abstraction skills** or **technical skills**. This barrier should be as small as possible, but the modeling tool shouldn't be simplified to the point of loosing the **rigor** and **consistency** that a meta-model can support.

These difficulties are easily felt when end-users are asked to collaborate in the design and implementation of a new DSL, for example.

5.2 Solution

Build a meta-model by providing examples at the model level.

Support META-MODELING BY EXAMPLE when users need to create a meta-model but it's more feasible to start by exploring options and experimenting at the model level than to keep models consistent with their meta-model at all times.

A model always needs an underlying meta-model, but the meta-model doesn't *have to* be explicitly captured before modeling is done — it may exist only in the modeler's mind. By relying on an explicit meta-meta-model, and making minimum assumptions about the meta-model level, modeling tools may support the creation of models in a very unconstrained way, and help modelers decide later what a compatible meta-model could look like.

On this later stage, the modeling tool can automatically infer a meta-model compatible with a given model or models, or it can use them to guide the creation or update of a meta-model when the user wishes to create/update it.

Although not within its main focus, the key principles of this pattern can also be applied between the *data* and *model* levels — a possible model can be inferred from information that has been structured in an *ad hoc* way, in the same way that a possible meta-model can be inferred from a given model.

This pattern can also be referred to as *Lazy Meta-Model* [Gabrysiak et al. 2011] in the sense that the meta-model needs only to be created when absolutely necessary, or as *Bottom-Up Meta-Modeling* [Sánchez-Cuadrado et al. 2012] in the sense that it encourages modeling to start from lower abstraction layers (i.e., the *bottom*).

5.3 Known Uses

Cho et al [Cho et al. 2011] tried to make the creation of Domain-Specific Modeling Languages more accessible to domain experts, by using examples of the language provided by the end-users themselves to infer the language's meta-model and semantics. This approach is referred to as *Modeling Language Creation By Demonstration*.

Kuhrmann has developed similar work [Kuhrmann 2011], striving to better support language engineers in the hard, time-consuming, and knowledge-intensive task of creating meta-models, models and DSLs. The *Process Development Environment* platform allows a free-form language design, and allows to visually represent domain entities and simple associations, that are used to derive a meta-model definition.

Smart Office Tools [Desmond et al. 2010] support a content model, capable of visually representing the domain knowledge as a domain diagram, but without tying the user to a specific meta-model. The user is able to customize the diagram notation by creating *styles*, which she can at any time annotate to formalize a meta-model.

5.4 Related Patterns

META-MODELING BY EXAMPLE always depends on the creation of a model, from which higher modeling levels are manually built or inferred. Such inference or creation of information constructs out of other pieces of information is something that this pattern has in common with FORMALIZATION. They are, otherwise, very different patterns, as FORMALIZATION acts only at the model level — it can be used when information is initially void of any explicit form of domain structure and only later is it made into an explicit model.

6. FORMALIZATION

If the modeler needs to change the meta-model and the modeling tool supports MODEL CO-EVOLUTION or META-MODELING BY EXAMPLE, she is able to introduce those changes without being constrained by the existing models. Other times, the modeler won't feel the need to change the meta-model but the instantiation link (i.e., the link between model and meta-model elements) may still be limiting in freely creating the model.

6.1 Problem

It is not always efficient, or even possible, to capture information as a model right from the beginning.

Modelers have various reasons for not using formal modeling tools, and rather often resource to free-form tools instead. Despite their **usability**, and advantages for **communication** and **creativity**, these tools don't offer any support for **maintaining** the results as models. Formal modeling tools provide such support, by enforcing conformance to a specific meta-model, and they confer the **semantics** needed to interpret the models objectively.

6.2 Solution

Derive formal/rigorous representations from information that was firstly captured informally/non-rigorously.

Support FORMALIZATION when the user's intent is to create a model that complies to a specific tool or to a well-known meta-model but she can't, or doesn't want, to be always constrained by that meta-model. FORMALIZATION frees the user from some or all meta-model constraints during modeling activities, easing the capture of the flow of thought or allowing to experiment and explore multiple options at the model level. The direct result of such a process, whilst not a model, can subsequently be formalized into a model.

FORMALIZATION always starts with free-form information. The move from free-form contents to a model can sometimes be done automatically by the modeling tool. For example, when sketching a diagram, the tool may store a combination of drawing gestures and a resulting raster image to infer what model elements the user meant to represent. The richer the information obtained through the users' input mechanisms, the easier may be to infer the semantics of what the user intended to express. Extraction techniques like image analysis or natural language processing may be used to support the move from free-form contents (e.g., a piece of text or a raster image) to a model. Additionally, when it's not possible to automatically infer a model, the modeling tool can interactively assist the user in manually building it from the contents.

6.3 Known Uses

The SKETCH API [Sangiorgi and Barbosa 2010] allows developers to add sketch recognition to modeling editors built for the Eclipse IDE. It leverages touch-enabled devices and their great potential as drawing tools, allowing to create and manipulate freehand sketches from which a formal model can be inferred and associated with an underlying meta-model.

Architects Workbench (AWB) [Abrams et al. 2006] provides totally free-form text entry and the ability to evolve them towards formally structured contents, maintaining the traceability from one form to the other. AWB's users can use a *markup and model* technique to create model elements directly from the text.

UNICASE [Helming et al. 2010] supports explicit traceability between information with different levels of abstraction and between loosely-formal *project models* (the artifacts that describe a software project, such as tasks, bug reports and informal communication) and *system models* (the artifacts that describe the system, such as functional requirements, UML models and detailed system specifications). This traceability information is then used to support the formalization process that underlies the propagation of changes from project models to system models.

6.4 Related Patterns

FORMALIZATION is similar to META-MODELING BY EXAMPLE to the extent that both patterns relax the constraints between the model and meta-model levels and help the modeler create new information constructs, or even automatically infer such information constructs, from other contents. Otherwise, they are very different patterns, as FORMALIZATION supports creating a model, and META-MODELING BY EXAMPLE supports introducing changes at both modeling levels — model and meta-model.

Organizing documentation towards DOMAIN-STRUCTURED INFORMATION [Correia et al. 2009] can be seen as a light attempt to FORMALIZATION, as the contents are organized according to their domain but not to the extend of becoming a model.

7. LINKED MODELS

USER-CRAFTED STATIC META-MODEL, MODEL CO-EVOLUTION and META-MODELING BY EXAMPLE allow modelers to increase their potential expressiveness at the model level by enriching the meta-model. Sometimes this means increasing the scope of the meta-model, which may make it difficult to manage or even overlap other readily available meta-models, that would be perfectly suited for modeling that part of the domain.

7.1 Problem

Models are conceived for specific domains, but modelers may need to address a broader domain than each model is able to address individually.

Wanting to be **expressive** in a certain domain, software developers sometimes resource to creating different models — for meta-models **specifically tailored** to different parts of that domain — or to providing different **views** over the same parts of the domain. However, these meta-models are not necessarily designed for **interoperability**, which implies a semantic gap between the produced models.

7.2 Solution

Allow two models or domain-specific languages to be linked as needed.

Support LINKED MODELS to enable users to connect different but complementary models in a flexible way. This pattern allows to compose models specialized to different parts of the domain.

Let the user of the modeling tool define, at the meta-model level, how the models can be linked or, instead, let her connect model elements in an *ad hoc* fashion (i.e., leave it entirely to the her choice which model elements can be connected, as appropriate). The best approach — meta-model-based or *ad hoc* — will depend on how the two domains relate to each other and on the amount of flexibility that the modeling tool is intended to provide. On both cases, this pattern may also be referred to as *Multiple Meta-Models*, in the sense that you may regard the result as a single set of connected model elements (i.e., a single *model*) that comply to more than one meta-model.

7.3 Known Uses

OMME (*Open Meta Modeling Environment*) [Volz and Jablonski 2010; Volz et al. 2011] is a meta-modeling environment implemented on top of the Eclipse platform that provides both textual and graphical notations. Among other features, this environment allows to represent and link models of arbitrary kind — like, for example, a process model and a data model — allowing to choose the models which fit best in a given situation.

Chiprianov et al propose a language tool for telecommunication network designers, that provides a partial syntactic and semantic automatic interoperability between different languages, corresponding to different viewpoints used in the definition of a telecommunication service [Chiprianov et al. 2010].

Microsoft Visio and similar tools allow users to choose between multiple *stencils*, thus allowing to combine modeling elements from multiple sets (i.e., multiple meta-models). In practice, the connections established using this approach arbitrarily link different models, while their meta-models remain independent. Despite its flexibility, the communication within a team using such models is sometimes more difficult than using completely meta-model driven models, as the semantics of the connections between these different elements is not defined by a meta-model, as highlighted by Gabrysiak et all [Gabrysiak et al. 2011].

7.4 Related Patterns

LINKED MODELS can be seen as a form of AUGMENTED MODELS were the contents used to augment the model are, in fact, other models or elements of other models.

When the meta-model doesn't establish how the models or model elements are to be linked, META-MODELING BY EXAMPLE can be used to allow modelers to explicitly create such rules at the meta-model level after links have been created.

Even though INFORMATION PROXIMITY [Correia et al. 2009] addresses software documentation in general, LINKED MODELS relies on the same principles, as does AUGMENTED MODELS.

8. AUGMENTED MODELS

Models hardly exist in isolation. Modelers may be able to LINK MODELS to go beyond the scope of a single meta-model, by connecting a model to other modeled contents. Sometimes, though, some of the contents that the modelers would like to relate a model to are not themselves a model.

8.1 Problem

Models are only a part of the information that needs to be captured and they need context to be fully understood.

Extending and tailoring the meta-model to the modeler's specific needs may be the most immediate solution when more expressiveness is needed at the model level, but supporting meta-model changes implies added complexity in the development of the modeling tool and the underlying meta-model. Modeling tools should be as **simple** as possible to develop, but also to be able to **express** as much detail as possible. Moreover, the information may be vague and difficult to represent as a model, if at all possible. Although we want information to be **structured** as much as possible — after all, the goal is to support the creation of models — we also want to support the capture of all **valuable** information independently of their level of structure. Free-form contents can contextualize, and thus allow to better **understand**, the created models.

8.2 Solution

Complement or annotate models with contents that provide additional context.

Support AUGMENTED MODELS to allow users to complement or annotate models, often with contents which are not structured or are loosely structured. The extra contents can be added at several granularity levels — they can be added to the model as a whole, or to specific model elements. In the later case, similarly to LINKED MODELS, the model elements to which new contents are added can be constrained at the meta-model level, or the modeling tool can allow them to be added arbitrarily to any model element.

Due to the lack of formal information constructs, it may be impossible to process the extra contents in any meaningful way, but they can be very useful as contextual information, required for the model to be better understood and be made use of.

8.3 Known Uses

Literate Modeling was born by taking the idea of Literate Programming beyond source-code. Instead of weaving only textual descriptions with source-code, they are also combined with UML models. In a way, models are *augmented* with textual descriptions that contextualize them and support their understanding by domain experts that are not proficient with modeling languages [Jobling 1993; Arlow et al. 1999].

The work by Breitman et al [Breitman et al. 2010] acknowledges that models are currently unable to accommodate all the levels of knowledge (from vague to concrete) that modelers need to represent, and proposes the creation of model *narratives*, as annotations in the form of graphics, audio, video, URLs, and other informal representations that are possibly vague and incomplete.

Nagel et al have proposed a solution to bridge the gap between informal communication and formal project model elements [Nagel et al. 2010]. They address the capture of audio conversions within their specific context, allowing to find specific information in a large number of recordings.

8.4 Related Patterns

AUGMENTED MODELS can be seen as a more general case of LINKED MODELS, were the contents being linked are not necessarily other models or elements of other models.

Model annotations can be free of any structural constraints, but it may be possible to employ FORMALIZATION to make their underlying structure explicit and make those contents part of the model itself. If this may result in new kinds of model elements, then META-MODELING BY EXAMPLE may too be used to define them at the meta-model level.

Even though INFORMATION PROXIMITY [Correia et al. 2009] addresses software documentation in general, AUGMENTED MODELS relies on the same principles, as does LINKED MODELS.

9. ACKNOWLEDGMENTS

We would like to thank Hironori Washizaki for his valuable feedback and support during the shepherding of this paper for PLoP 2013. We would also like to thank all the participants of our writer's workshop at PLoP, lead by Joshua Kerievski — Dibyendu Goswami, Jeffrey Overbey, Juan Reza and Russ Rubis. Finally, we would like to acknowledge both Fundação para a Ciência e Tecnologia and ParadigmaXis, which partially financed this work through the grant SFRH/BDE/33883/2009.

REFERENCES

ABRAMS, S., BLOOM, B., KEYSER, P., KIMELMAN, D., NELSON, E., NEUBERGER, W., ROTH, T., SIMMONDS, I., TANG, S., AND VLISSIDES, J. 2006. Architectural thinking and modeling with the architects' workbench. *IBM Systems Journal 45,* 3, 481–500.

ARLOW, J., EMMERICH, W., AND QUINN, J. 1999. Literate modelling - capturing business knowledge with the UML. In *The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, June 1998*, J. Bézivin and P.-A. Muller, Eds. Vol. 1618. Springer, Mulhouse, France, 189–199.

BREITMAN, K., PASTOR, O., AND BARBOSA, S. 2010. Flexible narrative representations: Bridging the gap between formal models and informal representations. In *ICSE 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010)*. Vol. 35. Cape Town, South Africa, 37–38.

CHIPRIANOV, V., KERMARREC, Y., AND ROUVRAIS, S. 2010. Meta-tools for software language engineering: a flexible collaborative modeling language for efficient telecommunications service design. In *ICSE 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010)*. Cape Town, South Africa.

CHO, H., SUN, Y., GRAY, J., AND WHITE, J. 2011. Key challenges for modeling language creation by demonstration. In *ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011)*. Waikiki, Hawaii, USA.

CICCHETTI, A., DI RUSCIO, D., ERAMO, R., AND PIERANTONIO, A. 2008. Automating co-evolution in model-driven engineering. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*. 222–231.

CORREIA, F. F. 2010. Supporting the evolution of software knowledge with adaptive software artifacts. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. SPLASH '10. ACM, New York, NY, USA, 231–232. ACM ID: 1869588.

CORREIA, F. F. 2013. Documenting software using adaptive software artifacts. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. SPLASH '13. ACM.

CORREIA, F. F., FERREIRA, H. S., FLORES, N., AND AGUIAR, A. 2009. Patterns for consistent software documentation. In *Proceedings of the Pattern Languages of Programs*. ACM, Chicago, Illinois, USA.

DESMOND, M., OSSHER, H., AND SIMMONDS, I. 2010. Towards smart office tools. In *SPLASH 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010@SPLASH)*. Reno, Nevada, USA.

FERREIRA, H. S., CORREIA, F. F., AND AGUIAR, A. 2009. Design for an adaptive object-model framework: An overview. In *4th Workshop on Models@run.time at MODELS 09*. 71–80.

FERREIRA, H. S., CORREIA, F. F., AGUIAR, A., AND FARIA, J. P. 2010. Adaptive object-models: a research roadmap. *IARIA Journal*.

FERREIRA, H. S., CORREIA, F. F., AND WELICKI, L. 2008. Patterns for data and metadata evolution in adaptive object-models. In *Proceedings of the 15th Conference on Pattern Languages of Programs*. ACM, Nashville, Tennessee, USA.

FERREIRA, H. S., CORREIA, F. F., YODER, J., AND AGUIAR, A. 2010. Core patterns of object-oriented meta-architectures. In *Proceedings of the 17th Conference on Pattern Languages of Programs*. ACM, Reno, Nevada, USA.

GABRYSIAK, G., GIESE, H., LÜDERS, A., AND SEIBEL, A. 2011. How can metamodels be used flexibly? In *ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011)*. Waikiki, Hawaii, USA.

GABRYSIAK, G., GIESE, H., AND SEIBEL, A. 2010. Using ontologies for flexibly specifying multi-user processes. In *ICSE 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010)*. Cape Town, South Africa.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*. Addison-Wesley Professional. Published: Hardcover.

HELMING, J., NARAYAN, N., ARNDT, H., KOEGEL, M., AND MAALEJ, W. 2010. From informal project management artifacts to formal system models. In *ICSE 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010)*. Cape Town, South Africa.

JOBLING, C. P. 1993. *The IEEE Grumman F14 Benchmark Problem - An Example of Literate Modelling in ACSL using noweb*.

KIMELMAN, D. AND HIRSCHMAN, K. 2010. Discussion and MindManager-Based structuring of workshop summary. http://www.ics.uci.edu/~nlopezgi/flexitools/presentations/SPLASH 2010 Workshop on Flexible Modeling Tools - v6.swf.

KIMELMAN, D. AND HIRSCHMAN, K. 2011. A spectrum of flexibility - lowering barriers to modeling tool adoption. In *ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011)*. Waikiki, Hawaii, USA.

KIMELMAN, D., OSSHER, H., VAN DER HOEK, A., AND STOREY, M.-A. 2010. SPLASH 2010 workshop on flexible modeling tools. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. SPLASH '10. ACM, New York, NY, USA, 283–284.

KUHRMANN, M. 2011. User assistance during domain-specific language design. In *ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011)*. Waikiki, Hawaii, USA.

NAGEL, M., HELMING, J., KOEGEL, M., AND NAUGHTON, H. 2010. Audio recording in software engineering. In *ICSE 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010)*. Cape Town, South Africa.

OMG – MOF. MetaObject Facility.

OSSHER, H., VAN DER HOEK, A., STOREY, M.-A., GRUNDY, J., AND BELLAMY, R. 2010. Flexible modeling tools (FlexiTools2010). In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*. ICSE '10. ACM, New York, NY, USA, 441–442.

OSSHER, H., VAN DER HOEK, A., STOREY, M.-A., GRUNDY, J., BELLAMY, R., AND PETRE, M. 2011. Workshop on flexible modeling tools (FlexiTools 2011). In *ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011)*. IEEE, 1192–1193.

SANGIORGI, U. B. AND BARBOSA, S. 2010. SKETCH: modeling using freehand drawing in eclipse graphical editors. In *ICSE 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010)*. Cape Town, South Africa.

STAHL, T. AND VOELTER, M. 2006. *Model-Driven Software Development: Technology, Engineering, Management* 1 Ed. Wiley.

SÁNCHEZ-CUADRADO, J., DE LARA, J., AND GUERRA, E. 2012. Bottom-up meta-modelling: An interactive approach. In *Model Driven Engineering Languages and Systems*. Springer, 3–19.

TOLVANEN, J.-P., POHJONEN, R., AND KELLY, S. 2007. Advanced tooling for domain-specific modeling: MetaEdit+. In *Sprinkle, J., Gray, J., Rossi, M., Tolvanen, JP (eds.) The 7th OOPSLA Workshop on Domain-Specific Modeling, Finland*.

VOLZ, B. AND JABLONSKI, S. 2010. OMME - a flexible modeling environment. In *SPLASH 2010 Workshop on Flexible Modeling Tools (FlexiTools 2010@SPLASH)*. Reno, Nevada, USA.

VOLZ, B., ZEISING, M., AND JABLONSKI, S. 2011. The open meta modeling environment. In *ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011)*. Waikiki, Hawaii, USA.

WACHSMUTH, G. 2007. Metamodel adaptation and model co-adaptation. In *ECOOP 2007–Object-Oriented Programming*. Springer, 600–624.